2.1 - The voting system and its software are implemented using trustworthy materials and best practices in software development.

2.1-# – Unstructured Control Flow

Application logic must contain no unstructured control constructs.

Deleted: SHALL

Applies to: Voting system software

Discussion

Although it is typically developed by the voting system manufacturer, border logic is constrained by the requirements of the third-party or COTS interface with which it must interact. It is not always possible for border logic to achieve its function while conforming to standard coding conventions. For this reason, border logic should be minimized relative to application logic and where possible, wrapped in a conforming interface. An example of border logic that could not be so wrapped is a customized boot manager that connects a bootable voting application to a COTS BIOS.

Status:	
Updated:	May 8, 2018
Source:	2007 6.4.1.5-B
Gap notes:	Software

2.1-#.1 – Goto

Arbitrary branches (a.k.a. gotos) are prohibited.

Applies to: Voting system software

Discussion

Status:	
Updated:	May 8, 2018
Source:	2007 6.4.1.5-B.1
Gap notes:	Software

2.1-#– Separation of code and data

Application logic must NOT compile or interpret configuration data or other input data as a programming language.

Deleted: SHALL

Applies to: Voting system software

Discussion

The applicable requirement in [VVSG2005] read "Operator intervention or logic that evaluates received or stored data must not re-direct program control within a program routine." That attempt to define what it means to compile or interpret data as a programming language caused confusion.

Deleted: shall

Distinguishing what is a programming language from what is not requires some professional judgment. However, in general, sequential execution of imperative instructions is a characteristic of conventional programming languages that should not be exhibited by configuration data. Configuration data must be declarative or informative in nature, not imperative.

For example: it is permissible for configuration data to contain a template that informs a report generating application as to the form and content of a report that it should generate, but it is not permissible for configuration data to contain instructions that are executed or interpreted to generate a report, essentially embedding the logic of the report generator inside the configuration data.
The reasons for this requirement are (1) mingling code and data is bad design, and (2) embedding logic within configuration data is an evasion of the conformity assessment process for application logic.

Status:	
Updated:	May 8, 2018
Source:	2007 6.4.1.5-C
Gap notes:	Software

2.3 - Voting system logic is clear, meaningful, and well-structured.

2.3-#– Acceptable programming languages

Application logic must be produced in a high-level programming language that has all of the following control constructs:

a. Sequence;

b. Loop with exit condition (e.g., for, while, and/or do-loops);

c. If/Then/Else conditional;

d. Case conditional; and

e. Block-structured exception handling (e.g., try/throw/catch).

Deleted: SHALL

Applies to: Voting system software

Discussion

The intent of this requirement is clarified in Part 1:6.4.1.5 “Structured programming” with discussion and examples of specific programming languages.

By excluding border logic, this requirement allows the use of assembly language for hardware-related segments, such as device controllers and handler programs. It also allows the use of an externally-imposed language for interacting with an Application Program Interface (API) or database query engine. However, the special code should be insulated from the bulk of the code, e.g. by wrapping it in callable units expressed in the prevailing language, to minimize the number of places that special code appears.

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.2-A
Gap notes: Software

Deleted: ¶
¶
¶

2.1-# – Hard-coded passwords

Voting system software must not contain hard-coded passwords.

Applies to: Voting system software

Discussion

Many examples of this vulnerability have previously been identified in voting system software. Additional information about this vulnerability can be found at CWE-259: Use of Hard-coded Password.

Status:
Updated:
Source: New
Gap notes: Software

2.1-# – Hard-coded cryptographic

Voting system software must not contain hard-coded cryptographic keys.

Applies to: Voting system software

Discussion

Many examples of this vulnerability have previously been identified in voting system software. Additional information about this vulnerability can be found at CWE-321: Use of Hard-coded Cryptographic Key.

Status:
Updated:
Source: New
Gap notes: Software

2.3-#.1– COTS language extensions are acceptable

The previous requirement (Acceptable programming languages) may be satisfied by using COTS extension packages to add missing control constructs to languages that could not otherwise conform.

Applies to: Voting system software

Discussion

For example, C99 [ISO99] does not support block-structured exception handling, but the construct can be retrofitted using (e.g.) [Sourceforge00] or another COTS package.

The use of non-COTS extension packages or manufacturer-specific code for this purpose is not acceptable, as it would place an unreasonable burden on the test lab to verify the soundness of an unproven extension (effectively a new programming language). The package must have a proven track record of performance supporting the assertion that it would be stable and suitable for use in voting systems, just as the compiler or interpreter for the base programming language must.

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.5-A.1
Gap notes: Software

2.3-#–Acceptable coding conventions

Application logic must adhere to a published, credible set of coding rules, conventions or standards (herein simply called "coding conventions") that enhance the workmanship, security, integrity, testability, and maintainability of applications.

Deleted: SHALL

Applies to: Voting system software

Discussion
Coding conventions that are excessively specialized or simply inadequate may be rejected on the grounds that they do not enhance one or more of workmanship, security, integrity, testability, and maintainability.

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.3-A
Gap notes: Software

2.3-#.1–Published coding conventions

Coding conventions must be considered published if and only if they appear in a publicly available book, magazine, journal, or new media with analogous circulation and availability, or if they are publicly available on the Internet.

Deleted: SHALL

Applies to: Voting system software

Discussion
This requirement attempts to clarify the "published, reviewed, and industry-accepted" language appearing in previous iterations of the VVSG, but the intent of the requirement is unchanged.
Following are examples of published coding conventions (links valid as of 2007-02). These are only examples and are not necessarily the best available for the purpose.
1. Ada: Christine Ausnit-Hood, Kent A. Johnson, Robert G. Pettit, IV, and Steven B. Opdahl, Eds., Ada 95 Quality and Style, Lecture Notes in Computer Science #1344, Springer-Verlag, 1995-06. Content available at http://www.iste.uni-stuttgart.de/ps/ada-doc/style_guide/cover.html and elsewhere.

2. C++: Mats Henricson and Erik Nyquist, Industrial Strength C++, Prentice-Hall, 1997. Content available at http://hem.passagen.se/erinyq/industrial/.
3. C#: "Design Guidelines for Class Library Developers," Microsoft. http://www.msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconnetframeworkdesignguidelines.asp.
4. Java: "Code Conventions for the Java™ Programming Language," Sun Microsystems. http://java.sun.com/docs/codeconv/.

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.3-A.1
Gap notes: Software

2.3-#.2–Credible

Coding conventions must be considered credible if and only if at least two different organizations with no ties to the creator of the rules or to the manufacturer seeking conformity assessment, and which are not themselves voting equipment manufacturers, independently decided to adopt them and made active use of them at some time within the three years before conformity assessment was first sought.

Deleted: SHALL

Applies to: Voting system software

Discussion
This requirement attempts to clarify the "published, reviewed, and industry-accepted" language appearing in previous iterations of the VVSG, but the intent of the requirement is unchanged.

Coding conventions evolve, and it is desirable for voting systems to be aligned with modern practices. If the "three year rule" was satisfied at the time that a system was first submitted for testing, it is considered satisfied for the purpose of subsequent reassessments of that system. However, new systems must meet the three year rule as of the time that they are first submitted for testing, even if they reuse parts of older systems.

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.3-A.2
Gap notes: Software

2.3-#–Header comments

Voting system software SHOULD include header comments that provide at least the following information for each callable unit (function, method, operation, subroutine, procedure, etc.):
a. The purpose of the unit and how it works (if not obvious);
b. A description of input parameters, outputs and return values, exceptions thrown, and side-effects;
c. Any protocols that must be observed (e.g., unit calling sequences);

d. File references by name and method of access (read, write, modify, append, etc.);
e. Global variables used (if applicable);
f. Audit event generation;
g. Date of creation; and
h. Change log (revision record).

Applies to: Voting system software

Discussion

Header comments and other commenting conventions should be specified by the selected coding conventions in a manner consistent with the idiom of the programming language chosen. If the coding conventions specify a coding style and commenting convention that make header comments redundant, then they may be omitted. Otherwise, in the event that the coding conventions fail to specify the content of header comments, the non-redundant portions of this generic guideline should be applied.

Change logs need not cover the nascent period, but they must go back as far as the first baseline or release that is submitted for testing, and should go back as far as the first baseline or release that is deemed reasonably coherent.

Status:	
Updated:	May 8, 2018
Source:	2007 6.4.1.6-A
Gap notes:	Software

2.4 - Voting system structure is modular, scalable, and robust.

2.4-#–Modularity

Application logic must be designed in a modular fashion.

Deleted: SHALL

Applies to: Voting system software

Discussion

See module. The modularity rules described here apply to the component submodules of a library.

Status:	
Updated:	May 8, 2018
Source:	2007 6.4.1.4-A
Gap notes:	Software

2.4-#.1–Module testability

Each module must have a specific function that can be tested and verified independently of the remainder of the code.

Deleted: SHALL

Applies to: Voting system software

Discussion

In practice, some additional modules (such as library modules) may be needed to compile the module under test, but the modular construction allows the supporting modules to be replaced by special test versions that support test objectives.

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.4-A.1
Gap notes: Software

2.4-#– Module size and identification

Modules must be small and easily identifiable.

Deleted: SHALL

Applies to: Voting system software

Discussion

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.4-B
Gap notes: Software

2.4-#.1– Callable unit length limit

No more than 50 % of all callable units (functions, methods, operations, subroutines, procedures, etc.) SHOULD exceed 25 lines of code in length, excluding comments, blank lines, and initializers for read-only lookup tables; no more than 5 % of all callable units SHOULD exceed 60 lines in length; and no callable units SHOULD exceed 180 lines in length.

Applies to: Voting system software

Discussion

"Lines," in this context, are defined as executable statements or flow control statements with suitable formatting.

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.4-B.1
Gap notes: Software

2.4-#.2– Lookup tables in separate files

Read-only lookup tables longer than 25 lines SHOULD be placed in separate files from other source code if the programming language permits it.

Applies to: Voting system software

Discussion

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.4-B.2
Gap notes: Software

2.4-#– Limiting cylcomatic complexity

The cyclomatic complexity measure of any given module must not exceed 15.

Applies to: Voting system software

Discussion

Cyclomatic complexity measures the amount of decision logic in a source code function. It is completely independent of text formatting and is nearly independent of programming language since the same fundamental decision structures are available and uniformly used in all procedural programming languages [NIST SP 500-235]. Modules with high cyclomatic complexity measures tend to be difficult to understand, test, and maintain. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication500-235.pdf

Status:
Updated:
Source: New
Gap notes: Software

2.5 - The voting system supports system processes and data with integrity.

2.5-#– Code coherency

Application logic must conform to the following subrequirements.

Deleted: SHALL

Applies to: Voting system software

Discussion

This is to scope the following subrequirements to application logic. For COTS software where source code is unobtainable, they would be unverifiable.

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.7-A
Gap notes: Software

2.5-#.1– Self-modifying code

Self-modifying code is prohibited.

Applies to: Voting system software

Discussion

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.7-A.1
Gap notes: Software

2.5-#.2– Unsafe concurrency

Application logic must be free of race conditions, deadlocks, livelocks, and resource starvation.

Deleted: SHALL

Applies to: Voting system software

Discussion

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.7-A.2
Gap notes: Software

2.5-#.3–Code integrity, no strange compilers

If compiled code is used, it must only be compiled using a COTS compiler.

Deleted: SHALL

Applies to: Voting system software

Discussion

This prohibits the use of arbitrary, nonstandard compilers and consequently the invention of new programming languages.

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.7-A.3
Gap notes: Software

2.5-#.4–Interpreted code, specific COTS interpreter

If interpreted code is used, it must only be run under a specific, identified version of a COTS runtime interpreter.

Deleted: SHALL

Applies to: Voting system software

Discussion

This ensures that (1) no arbitrary, nonstandard interpreted languages are used, and (2) the software tested and approved during the conformity assessment process does not change behavior because of a change to the interpreter.

Status:	
Updated:	May 8, 2018
Source:	2007 6.4.1.7-A.4
Gap notes:	Software

2.5-#– Prevent tampering with code

Programmed devices must prevent replacement or modification of executable or interpreted code (e.g., by other programs on the system, by people physically replacing the memory or medium containing the code, or by faulty code) except where this access is necessary to conduct the voting process.

Deleted: SHALL

Applies to: Voting system software

Discussion

This requirement may be partially satisfied through a combination of read-only memory (ROM), the memory protection implemented by most popular COTS operating systems, error checking as described in Part 1:6.4.1.8 “Error checking”, and access and integrity controls.

Status:	
Updated:	May 8, 2018
Source:	2007 6.4.1.7-B
Gap notes:	Software

2.5-#– Prevent tampering with data

All voting devices must prevent access to or manipulation of configuration data, vote data, or audit records (e.g., by physical tampering with the medium or mechanism containing the data, by other programs on the system, or by faulty code) except where this access is necessary to conduct the voting process.

Deleted: SHALL

Applies to: Voting system software

Discussion

This requirement may be partially satisfied through a combination of the memory protection implemented by most popular COTS operating systems, error checking as described in Part 1:6.4.1.8 “Error checking”, and access and integrity controls. Systems using mechanical counters to store vote data must protect the counters from tampering. If vote data are stored on paper, the paper must be protected from tampering. Modification of audit records after they are created is never necessary.

Status:	
Updated:	May 8, 2018

Source: 2007 6.4.1.7-C
Gap notes: Software

2.5-#– Monitor I/O errors

Programmed devices must provide the capability to monitor the transfer quality of I/O operations, reporting the number and types of errors that occur and how they were corrected.

Deleted: SHALL

Applies to: Voting system software

Discussion

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.7-D
Gap notes: Software

2.5-#– Detect garbage input

Programmed devices must check information inputs for completeness and validity.

Deleted: shall

Applies to: Voting system software

Deleted:

Discussion

This general requirement applies to all programmed devices, while the specific ones following are only enforceable for application logic.

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.8-A
Gap notes: Software

2.5-#.1–Defend against garbage input

Programmed devices must ensure that incomplete or invalid inputs do not lead to irreversible error.

Deleted: SHALL

Applies to: Voting system software

Deleted:

Discussion

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.8-A.1
Gap notes: Software

2.5-#.2–Validate and Filter Input

The voting system must validate all input against expected parameters, such as data presence, length, type, format, uniqueness, or inclusion in a set of whitelisted values

Applies to: Voting system software

Discussion
Input includes data from any input source: input devices (touchscreens, keyboards, keypads, optical/digital scanners, assistive devices, etc.), networking port, data port, or file.

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.8-A.1
Gap notes: Software

2.5-#– Proper escaping and encoding of output

Software output must be properly encoded and escaped.

Applies to: Voting system software

Discussion
The output of a software module can be manipulated and/or abused by attackers in unexpected ways to perform malicious actions. Ensuring that outputted data is of an expected type or format assists in preventing this abuse. Additional information about this software weakness can be viewed in within CWE 116: Improper Encoding or Escaping of Output.

Status:
Updated:
Source: New
Gap notes: Software

2.5-#.1– Satanize output

The voting system must sanitize all output to remove or neutralize the effects of any escape characters, control signals, or scripts contained in the data which could adversely manipulate the output source.

Applies to: Voting system software

Discussion
Output includes data to any output source: output devices (touchscreens, LCD screens, printers, assistive devices, etc.), networking port, data port, or file. Applies to all parts of the voting system including the EMS.

Status:
Updated:

Source: New
Gap notes: Software

2.5-#.2– Stored injection

The voting system must sanitize all output to files and databases to remove or neutralize the effects of any escape characters, control signals, or scripts contained in the data which could adversely manipulate the voting system if the stored data is read or imported at a later date or by another part of the voting system.

Applies to: Voting system software

Discussion
A stored injection attack saves malicious data which is harmless when stored, but which is potent when read later in a different context or when converted to a different format. For example, a malicious script might be written to a file and do no harm to the voting machine, but later be evaluated and harmful when the file is transferred and read by the EMS. Input should also be filtered, but sanitizing stored output provides defense in depth. Examples of stored injection include

Status:
Updated:
Source: New
Gap notes: Software

2.5-#– Mandatory internal error checking

Application logic that is vulnerable to the following types of errors must check for these errors at run time and respond defensively (as specified by Requirement Part 1:6.4.1.8-F) when they occur:

a. Common memory management errors, such as out-of-bounds accesses of arrays, strings, and buffers used to manage data;
b. Uncontrolled format strings;
c. CPU-level exceptions such as address and bus errors, dividing by zero, and the like;
d. Variables that are not appropriately handled when out of expected boundaries;
e. Numeric and integer overflows; or
f. Validation of array indices;
f. Known programming language specific vulnerabilities.

Deleted: SHALL

Deleted: O

Deleted: or

Deleted: (includes

Deleted: move

Deleted:)

Deleted: Stack overflow errors

Applies to: Voting system software

Discussion
It is acceptable, even expected, that logic verification will show that some error checks cannot logically be triggered and some exception handlers cannot logically be invoked. These checks and exception handlers are not redundant – they provide defense-in-depth against faults that escape detection during logic verification.

Deleted:

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.8-B
Gap notes: Software

2.5-#.1– Array overflows

If the application logic uses arrays, vectors, or any analogous data structures and the programming language does not provide automatic run-time range checking of the indices, the indices must be ranged-checked on every access.

Deleted: SHALL

Applies to: Voting system software

Discussion

Range checking code should not be duplicated before each access. Clean implementation approaches include:

1. Consistently using dedicated accessors (functions, methods, operations, subroutines, procedures, etc.) that range-check the indices;
2. Defining and consistently using a new data type or class that encapsulates the range-checking logic;
3. Declaring the array using a template that causes all accessors to be range-checked; or
4. Declaring the array index to be a data type whose enforced range is matched to the size of the array.

Range-enforced data types or classes may be provided by the programming environment or they may be defined in application logic. If acceptable values of the index do not form a contiguous range, a map structure may be more appropriate than a vector.

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.8-B.1
Gap notes: Software

2.5-#.2– Buffer overflows

Deleted: Stack overflows

If an overflow does not automatically result in an exception, the application logic must explicitly check for and prevent the overflow.

Deleted: stack

Deleted: SHALL

Deleted: stack

Applies to: Voting system software

Discussion

Embedded system developers use a variety of techniques for avoiding stack overflow. Commonly, the stack is monitored and warnings and exceptions are thrown when thresholds are crossed. In non-embedded contexts, stack overflow often manifests as a CPU-level exception related to memory segmentation, in which case it can be handled pursuant to Requirement Part 1:6.4.1.8-B.3 and Requirement Part 1:6.4.1.9-D.2. Types of software weakness covered under this requirement include CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow'), CWE-121: Stack-based Buffer Overflow, and CWE-122: Heap-based Buffer Overflow.

Formatted: Highlight

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.8-B.2
Gap notes: Software

2.5-#.3– CPU traps

The application logic must implement such handlers as are needed to detect and respond to CPU-level exceptions.

Deleted: SHALL

Applies to: Voting system software

Discussion
For example, under Unix a CPU-level exception would manifest as a signal, so a signal handler is needed. If the platform supports it, it is preferable to translate CPU-level exceptions into software-level exceptions so that all exceptions can be handled in a consistent fashion within the voting application; however, not all platforms support it.

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.8-B.3
Gap notes: Software

Deleted: ¶

2.5-#.4– Garbage input parameters

All scalar or enumerated type parameters whose valid ranges as used in a callable unit (function, method, operation, subroutine, procedure, etc.) do not cover the entire ranges of their declared data types must be range-checked on entry to the unit.

Deleted: SHALL

Applies to: Voting system software

Discussion
This applies to parameters of numeric types, character types, temporal types, and any other types for which the concept of range is well-defined.[7] In cases where the restricted range is frequently used and/or associated with a meaningful concept within the scope of the application, the best approach is to define a new class or data type that encapsulates the range restriction, eliminating the need for range checks on each use.

This requirement differs from Requirement Part 1:6.4.1.8-A, which deals with user input that is expected to contain errors, while this requirement deals with program internal parameters, which are expected to conform to the expectations of the designer. User input errors are a normal occurrence; the errors discussed here are grounds for throwing exceptions.

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.8-B.4

Gap notes: Software

2.5-#.5– Numeric overflows

If the programming language does not provide automatic run-time detection of numeric overflow, all arithmetic operations that could potentially overflow the relevant data type must be checked for overflow.

Deleted: SHALL

Applies to: Voting system software

Discussion

This requirement should be approached in a manner similar to Requirement Part 1:6.4.1.8-B.1. Overflow checking should be encapsulated as much as possible.

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.8-B.5
Gap notes: Software

2.5-#.6 – Uncontrolled Format Strings

Voting system software must not contain uncontrolled format strings.

Applies to: Voting system software

Discussion

Many examples of this vulnerability have previously been identified in voting system software. Additional information about this vulnerability can be found at CWE 134: Use of Externally-Controlled Format String.

Status:
Updated:
Source: New
Gap notes: Software

2.5-#– Recommended internal error checking

Application logic that is vulnerable to the following types of errors must check for these errors at run time and respond defensively (as specified by Requirement Part 1:6.4.1.8-F) when they occur.

Deleted: SHOULD

a. Pointer variable errors; and
b. Dynamic memory allocation and management errors

Applies to: Voting system software

Discussion

Status:	
Updated:	May 8, 2018
Source:	2007 6.4.1.8-C
Gap notes:	Software

2.5-#.1– Pointers

If application logic uses pointers or a similar mechanism for specifying absolute memory locations, the application logic must validate pointers or addresses before they are used.

Deleted: SHOULD

Applies to: Voting system software

Discussion

Improper overwriting should be prevented in general as required by Requirement Part 1:6.4.1.7-B and Requirement Part 1:6.4.1.7-C. Nevertheless, even if read-only memory would prevent the overwrite from succeeding, an attempted overwrite indicates a logic fault that must be corrected.

Pointer use that is fully encapsulated within a standard platform library is treated as COTS software.

Status:	
Updated:	May 8, 2018
Source:	2007 6.4.1.8-C.1
Gap notes:	Software

2.5-#– Memory mismanagement

If dynamic memory allocation is performed in application logic, the application logic must be able to be instrumented and/or analyzed with a COTS tool for detecting memory management errors.

Deleted: SHOULD

Applies to: Voting system software

Discussion

Dynamic memory allocation that is fully encapsulated within a standard platform library is treated as COTS software. This is "should" not "must" only because such tooling may not be available or applicable in all cases. See [Valgrind07] discussion of supported platforms and the barriers to portability.

Deleted: shall

Status:	
Updated:	May 8, 2018
Source:	2007 6.4.1.8-D
Gap notes:	Software

2.5-#– Nullify freed pointers

If pointers are used, any pointer variables that remain within scope after the memory they point to is deallocated must be set to null or marked as invalid (pursuant to the idiom of the programming language used) after the memory they point to is deallocated.

Deleted: SHALL

Applies to: Voting system software

Discussion

If this is not done automatically by the programming environment, a callable unit should be dedicated to the task of deallocating memory and nullifying pointers. Equivalently, "smart pointers" like the C++ std::auto_ptr can be used to avoid the problem. One should not add assignments after every deallocation in the source code.

In languages using garbage collection, memory is not deallocated until all pointers to it have gone out of scope, so this requirement is moot.

Status:	
Updated:	May 8, 2018
Source:	2007 6.4.1.8-E
Gap notes:	Software

2.5-#– React to errors detected

The detection of any of the errors enumerated in Requirement Part 1:6.4.1.8-B and Requirement Part 1:6.4.1.8-C must be treated as a complete failure of the callable unit in which the error was detected. An appropriate exception must be thrown and control must pass out of the unit forthwith.

Deleted: SHALL

Deleted: SHALL

Deleted: SHALL

Applies to: Voting system

Discussion

Status:	
Updated:	May 8, 2018
Source:	2007 6.4.1.8-F
Gap notes:	Software

2.5-#– Do not disable error checks

Error checks detailed in Requirement Part 1:6.4.1.8-B and Requirement Part 1:6.4.1.8-C must remain active in production code.

Deleted: SHALL

Applies to: Voting system

Discussion

These errors are incompatible with voting integrity, so masking them is unacceptable.

Manufacturers should not implement error checks using the C/C++ assert() macro. It is often disabled, sometimes automatically, when software is compiled in production mode. Furthermore, it does not appropriately throw an exception, but instead aborts the program.

"Inevitably, the programmed validity checks of the defensive programming approach will result in run-time overheads and, where performance demands are critical, many checks are often removed from the operational software; their use is restricted to the testing phase where they can identify the misuse of components by faulty designs. In the context of producing complex systems which can never be fully tested, this tendency to remove the protection afforded by programmed validity checks is most regrettable and is not recommended here." [Moulding89]

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.8-G
Gap notes: Software

2.5-#– Election integrity monitoring

To the extent possible, electronic devices must proactively detect or prevent basic violations of election integrity (e.g., stuffing of the ballot box or the accumulation of negative votes) and alert an election official or administrator if they occur.

Commented [FJM(1]: Does this belong in detection and monitoring or system integrity?

Deleted: SHALL

Applies to: Voting system

Discussion

Equipment can only verify those conditions that are within the scope of what the equipment does. However, insofar as the equipment can detect something that is blatantly wrong, it should do so and raise the alarm. This provides defense-in-depth to supplement procedural controls and auditing practices.

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.8-K
Gap notes:

Deleted: ¶

2.5-#– SQL injection

The voting system application must defend against SQL injection.

Applies to: Voting system software

Discussion

SQL injection is a classic type of software weakness still prevalent today. SQL injection is not just a web-based issue as any application accepting untrusted user input and passing it to a database can be vulnerable. Additional information about this software weakness can be viewed in within CWE 89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection').

Status:
Updated:
Source: New
Gap notes: Software

2.5-#.1– Parameterized queries

Any structured statement or command being prepared using dynamic data (including user input) to be sent to a database or other process must parameterize the data inputs and apply strict type casting and content filters on the data (e.g., prepared statements).

Applies to: Voting system software

Discussion
Parametrized queries are a common defense against this class of software weakness.

Status:
Updated:
Source: New
Gap notes: Software

2.6 - The voting system handles errors robustly and gracefully recovers from failure.

2.6-#– Block-structured exception handling

Application logic must handle exceptions using block-structured exception handling constructs.

Deleted: SHALL

Applies to: Voting system software

Discussion
Block-structured exception handling," is the ability to associate exception handlers with blocks of logic, and implicitly, the presence of the exception concept in the programming language. (This simply means try/throw/catch or equivalent statements, and should not be confused with the specific implementation known as Structured Exception Handling (SEH) [Pietrek97].[2]) Unlike deeply nested blocks, exceptions cannot be eliminated by restructuring logic. "When exceptions are not used, the errors cannot be handled but their existence is not avoided." [ISO00a]

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.5-A

Gap notes:	Software

2.6-#.1– Legacy library units must be wrapped

If application logic makes use of any COTS or that do not throw exceptions when exceptional conditions occur, those units must be wrapped in callable units that check for the relevant error conditions and translate them into exceptions, and the remainder of application logic must use only the wrapped version.

Deleted: SHALL

Deleted: SHALL

Applies to: Voting system software

Discussion

For example, if an application written in C99 [ISO99] + cexcept [Sourceforge00] used the malloc function of libc, which returns a null pointer in case of failure instead of throwing an exception, the malloc function would need to be wrapped. Here is one possible implementation:

```
void *checkedMalloc (size_t size) {
        void *ptr = malloc (size);
        if (!ptr)
                Throw bad_alloc;
        return ptr;
}
#define malloc checkedMalloc
```

Wrapping legacy functions avoids the need to check for errors after every invocation, which both obfuscates the application logic and creates a high likelihood that some or many possible errors will not be checked for.

In C++, it would be preferable to use one of the newer mechanisms that already throw exceptions on failure and avoid use of legacy functions altogether.

Status:
Updated:	May 8, 2018
Source:	2007 6.4.1.5-A.1
Gap notes:	Software

2.6-#– Intentional exceptions

Exceptions must only be used for abnormal conditions. Exceptions must NOT be used to redirect the flow of control in normal ("non-exceptional") conditions.

Deleted: SHALL

Deleted: SHALL

Applies to: Voting system software

Discussion

"Intentional exceptions" cannot be used as a substitute for arbitrary branch. Normal, expected events, such as reaching the end of a file that is being read from beginning to end or receiving invalid input from a user interface, are not exceptional conditions and should not be implemented using exception handlers.

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.5-B.2
Gap notes: Software

2.6-#.1– Unstructured exception handling

Unstructured exception handling (e.g., On Error GoTo, setjmp/longjmp, or explicit tests for error conditions after every executable statement) is prohibited.

Applies to: Voting system software

Discussion

The internal use of such constructs by a COTS extension package that adds block-structured exception handling to a programming language that otherwise would not have it, as described in Requirement Part 1:6.4.1.2-A.1, is allowed. Analogously, it is not a problem that source code written in a high-level programming language is compiled into low-level machine code that contains arbitrary branches. It is only the direct use of low-level constructs in application logic that presents a problem.

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.5-B.3
Gap notes: Software

2.6-#– System must survive device failure

All systems must be capable of resuming normal operation following the correction of a failure in any device.

Deleted: shall

Deleted: SHALL

Applies to: Voting system software

Discussion

Status:
Updated: May 8, 2018
Source: 2007 6.4.1.9-A
Gap notes: Software

2.6-#– Failures must not compromise voting or audit data

Exceptions and system recovery must be handled in a manner that protects the integrity of all recorded votes and audit log information.

Deleted: shall

Deleted: SHALL

Applies to: Voting system software

Discussion

Status:	
Updated:	May 8, 2018
Source:	2007 6.4.1.9-A
Gap notes:	Software

2.6-#– Device must survive component failure

Deleted: shall

All voting devices must be capable of resuming normal operation following the correction of a failure in any component (e.g., memory, CPU, ballot reader, printer) provided that catastrophic electrical or mechanical damage has not occurred.

Deleted: SHALL

Applies to: Voting system software

Discussion

Status:	
Updated:	May 8, 2018
Source:	2007 6.4.1.9-C
Gap notes:	Software

2.6-#– Controlled recovery

Error conditions must be corrected in a controlled fashion so that system status may be restored to the initial state existing before the error occurred.

Deleted: SHALL

Applies to: Voting system software

Discussion

"Initial state" refers to the state existing at the start of a logical transaction or operation. Transaction boundaries must be defined in a conscientious fashion to minimize the damage. Language changed to "may" because election officials responding to the error condition might want the opportunity to select a different state (e.g., controlled shutdown with memory dump for later analysis).

Status:	
Updated:	May 8, 2018
Source:	2007 6.4.1.9-D
Gap notes:	Software

2.6-#.1– Nested error conditions

Nested error conditions that are corrected without reset, restart, reboot, or shutdown of the voting device must be corrected in a controlled sequence so that system status may be restored to the initial state existing before the first error occurred.

Deleted: SHALL

Applies to: Voting system software

Discussion

Status:	
Updated:	May 8, 2018
Source:	2007 6.4.1.9-D.1
Gap notes:	Software

2.6-#.2– Reset CPU error states

CPU-level exceptions that are corrected without reset, restart, reboot, or shutdown of the voting device must be handled in a manner that restores the CPU to a normal state and allows the system to log the event and recover as with a software-level exception.

Deleted: SHALL

Applies to: Voting system software

Discussion

System developers should test to see how CPU-level exceptions are handled and make any changes necessary to ensure robust recovery. Invocation of any other error routine while the CPU is in an exception handling state is to be avoided – software error handlers often do not operate as intended when the CPU is in an exception handling state.

If the platform supports it, it is preferable to translate CPU-level exceptions into software-level exceptions so that all exceptions can be handled in a consistent fashion within the voting application; however, not all platforms support it.

Status:	
Updated:	May 8, 2018
Source:	2007 6.4.1.9-D.2
Gap notes:	Software

2.6-#– Coherent checkpoints

When recovering from non-catastrophic failure of a device or from any error or malfunction that is within the operator's ability to correct, the system must restore the device to the operating condition existing immediately prior to the error or failure, without loss or corruption of voting data previously stored in the device.

Deleted: SHALL

Applies to: Voting system software

Discussion

If, as discussed in Requirement Part 1:6.4.1.9-D, the system is left in something other than the last known good state for diagnostic reasons, this requirement clarifies that it must revert to the last known good state before being placed back into service.

Status:	
Updated:	May 8, 2018
Source:	2007 6.4.1.9-E
Gap notes:	Software